POLITECNICO
MILANO 1863

RASD & DD project discussion

2 February 2024

Out line

Requirements analysis and specification document:

- Overall description
- Specific requirements
- Formal analysis using Alloy

Design document:

- Architectural design
- User Interface design
- Implementation integration and test plan

Overall description: goal of the system and its boundaries

Goals of the system:

1. Educators can create and manage a tournament.
2. Educators can create new badges within a tournament.
3. Educators can create a battle within a tournament to which they have access to.
4. Students can subscribe to a tournament within the specified deadline.
5. Students can enroll in a battle within their tournaments.
6. Students can form a team by invite.
7. Students can see their team score in a battle. ←
8. Users can see the list of ongoing tournaments.
9. Users can see all tournaments ranking.
10. Users can visualize collected badges of students by visiting their profile.

Boundaries:

- The educators autonomously ideate tournaments and battles and publish them through the platform. Similarly, students choose tournaments and battles that they want to participate to and autonomously form teams and write a solution.
- The systems is responsible for notifying all users about incoming tournaments, battles, results, for displaying all necessary information, for enabling educators to manage tournaments and battles and students to manage their teams.

Specific requirements: most important requirements and domain assumption

Requirements:

1. The system shall allow the educators to create tournaments.
2. The system shall allow the educators to manage their tournaments, in particular invites other collaborators and ends the tournament.
3. The system shall allow students to subscribe in tournaments.
4. The system shall allow students to enroll in a battle within a tournament.
5. The system shall automatically evaluate submissions. ⟵

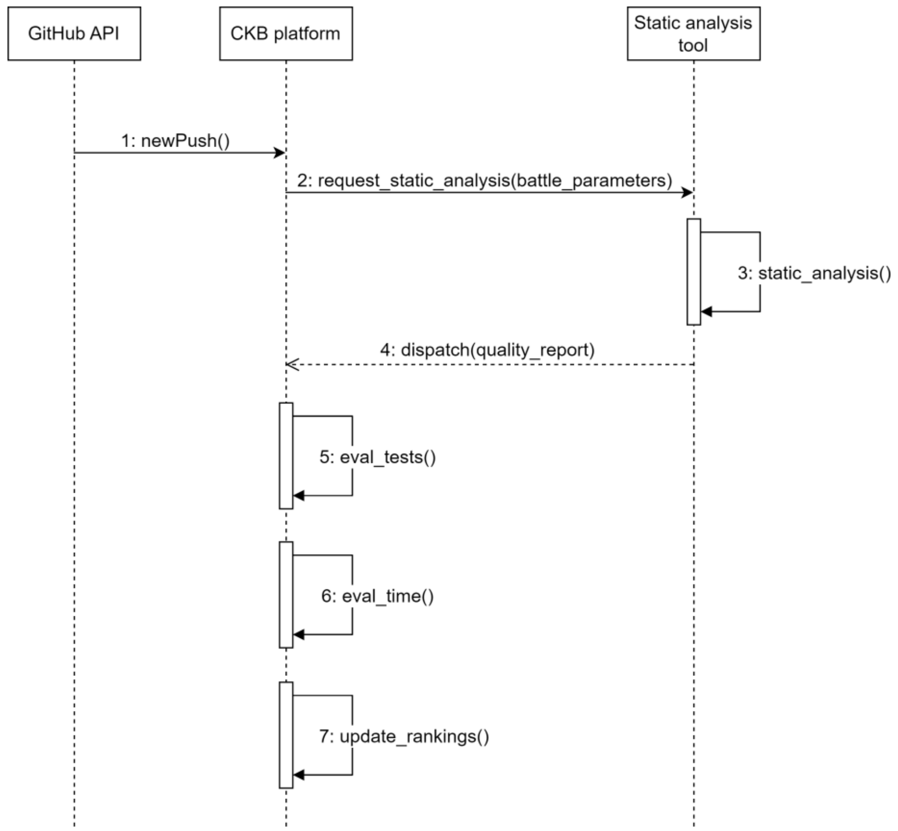
Domain assumption:

1. Educators must be qualified and experienced in software development and related fields.
2. Each team must fork the GitHub repository created by the CKB platform once and set up an automated workflow through GitHub Actions.
3. The automatic workflow correctly trigger the CKB platform in order to evaluate the new push. ⟵
4. The static analysis tools used must be reliable and perform correct evaluations. ⟵

Specific requirements: an important use case

UC.16 Perform automatic evaluation

Name	Perform automatic evaluation.
Actors	GitHub API, static analysis tool.
Entry conditions	A team push a new solution on the GitHub repository.
Event flow	1. The GitHub API triggers the CKB platform. 2. The system request the quality report based on the parameter chosen at battle creation. 3. The static analysis tool gives back the quality report. 4. The system performs functional evaluation based on tests cases. 5. The system performs time evaluation. 6. The system updates the real-time battle ranking and the student's tournament ranking.
Exit conditions	The real-time rank and the student tournament ranking is updated.

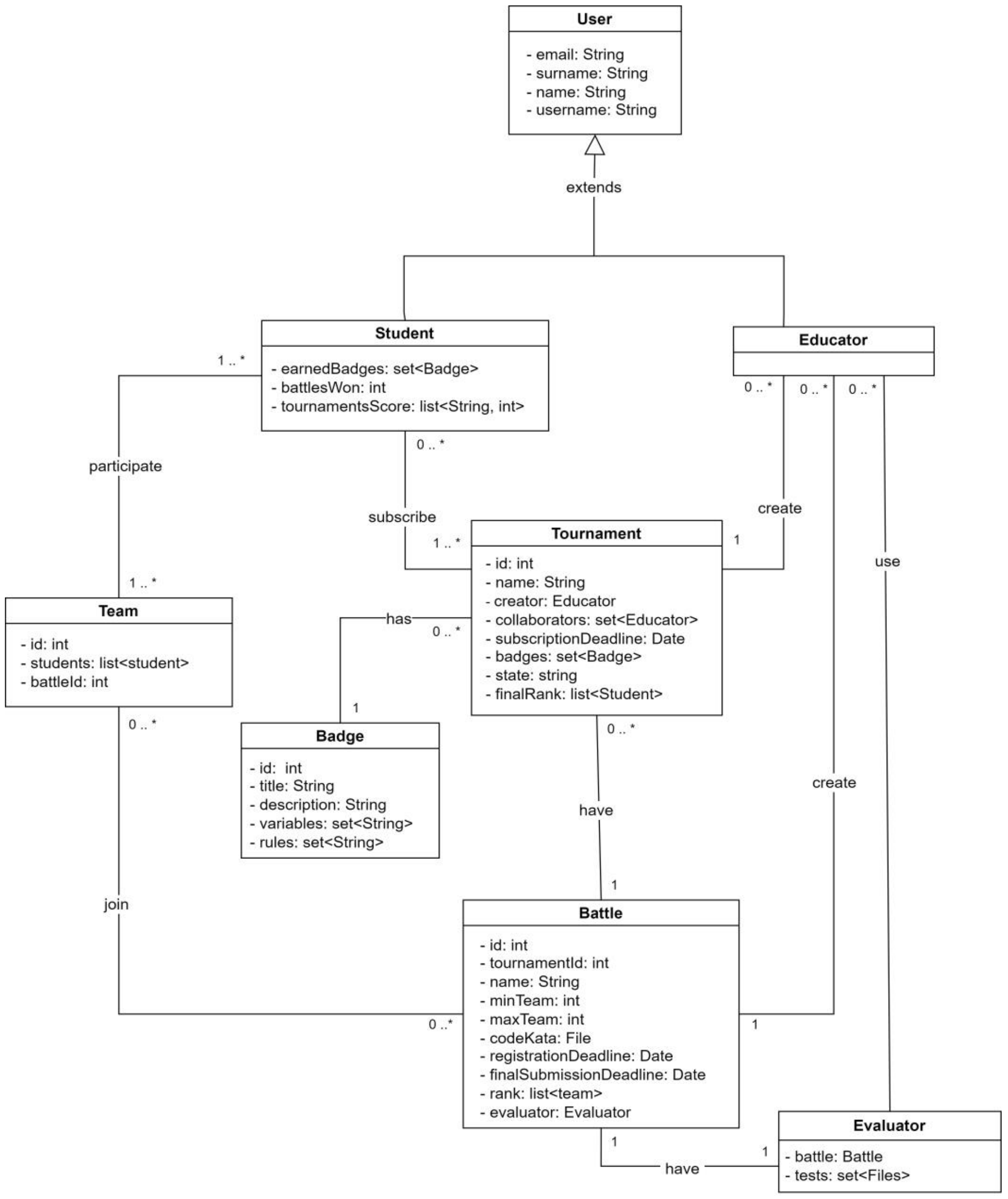


Our domain class diagram

We have decided to model through Alloy the structure defined in the Domain Class Diagram to ensure the coherence of our model.

```
sig Battle{
      battleId : BattleId,
      tournamentId : TournamentId,
  creator : Educator,
  code : one CodeKata,
  participants : set Student,
  teams : set Team,
  maxTeamSize : Int,
  minTeamSize : Int,
  registrationDeadline: Date,
  submissionDeadline: Date,
  state : one State,
  battleRanking : set BattleScore
}
{
  maxTeamSize <=4
  minTeamSize > 0
  maxTeamSize >= minTeamSize
}

sig Tournament {
      tournamentId : TournamentId,
  creator : one Educator,
  collaborators : set Educator,
  battles : set Battle,
  participants : set Student,
  subscriptionDeadline : Date,
  badges : set Badge,
  state : one State,
  tournamentRanking : set TournamentScore
}
```



Formal analysis using Alloy: the worlds

In detail we aimed to represent three main aspects:

- Verify the structure about the existence of a tournament, a battle and the presence of students in them.

```
//ensures that if a student is enrolled in a battle then he/she
    is a participant of the tournament
fact ifStudentInBattleThenInTournament{
  all t : Tournament, b : t.battles, s : Student| s in b.
   participants => s in t.participants
}
```

- Verify the behavior and the presence of the leaderboards of battles and tournaments.

```
//ensures that the ranking of a tournament is only composed by
    participants of the tournament
fact NoStudentEnrolledWithoutTScore{
  all t : Tournament, s : t.participants | one tlt :t.
    tournamentRanking.student|s = tlt
}
```

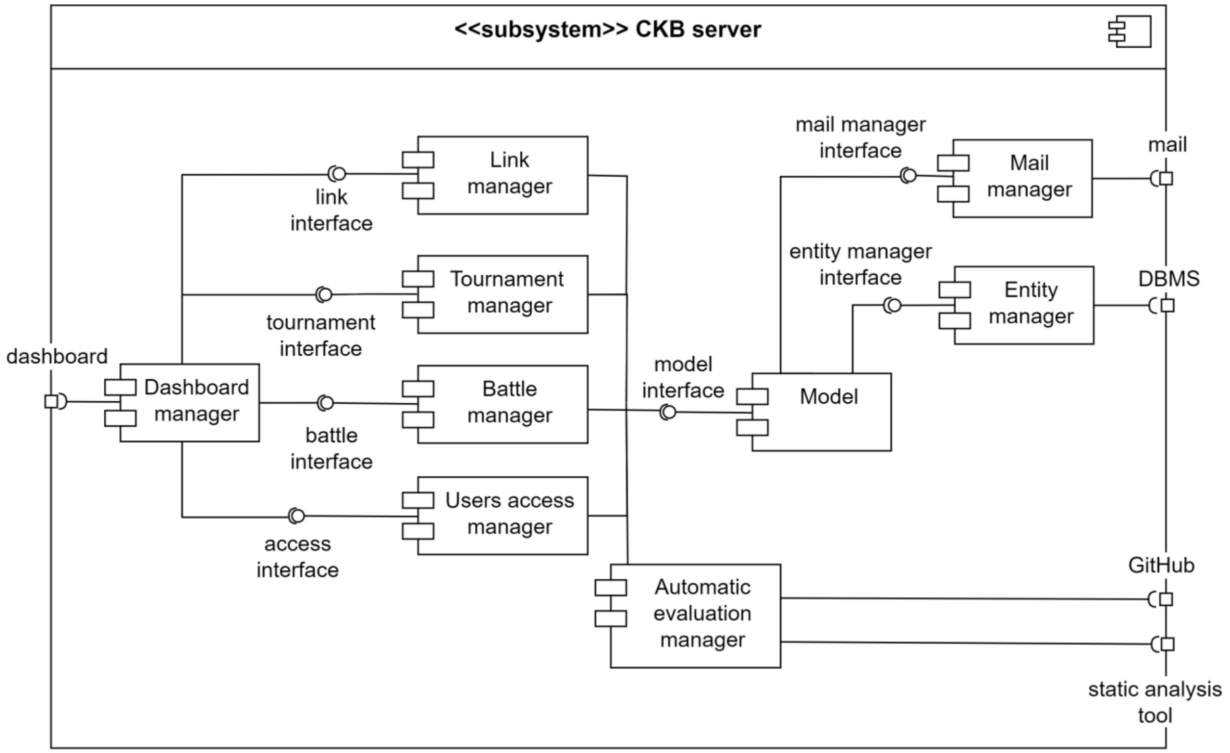
- *Verify* the *coherence about the presence of badges and their assignment.*

```
//ensures that a student, who is not a participant of a
    tournament, cannot collects the tournament's badges
fact StudentThatAreNotInATournamentCannotHaveItsBadges{
  all s : Student, t : Tournament |
    (s not in t.participants) implies not (one b : Badge | b in
    s.collectedBadges and b in t.badges)
}
```

Architectural Design: Component View

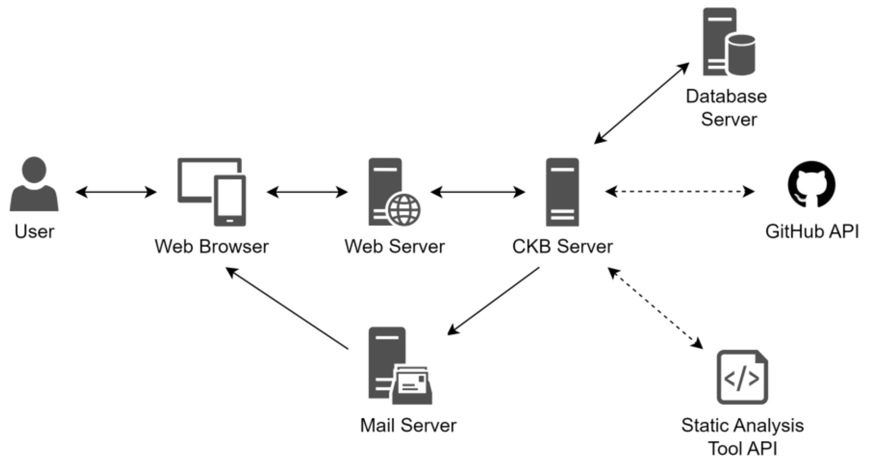
Now let’s see how the different components interact with each other and with the user:

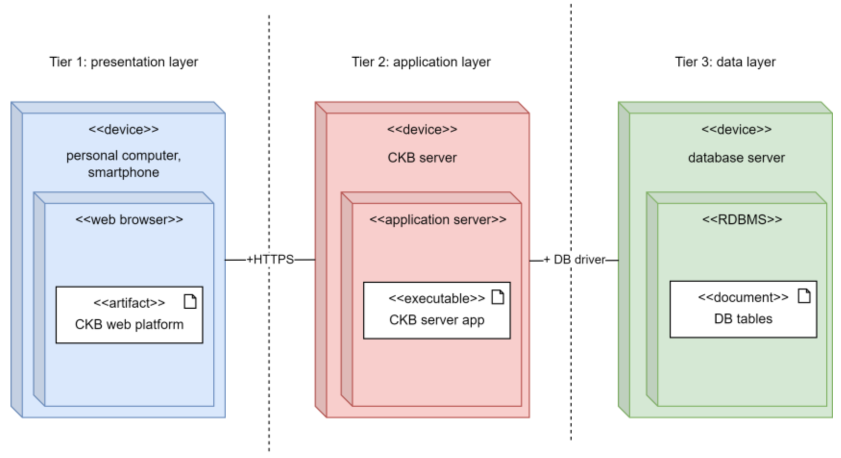
- Dashboard offers the user interface and adapt what the user see, using the internal interfaces of link, tournament, battle and user.
- The four internal modules will interact with the model to modify it or to get information depending on the pending request.
- Finally, the automatic evaluation manager gathers the data from the GitHub API, collects the analysis from the automatic evaluation tools and sends the results to the model.



Architectural design: styles, paradigm and meaningful interactions

- Client-Server: the application works with a client server interaction trough messages and requests. The server is responsive to client requests.
- 3-Tiers: we organize the system into three logical and physical layers with a thin client configuration. The three layers are: Presentation layer, Application layer ,Data layer
- MVC: Model (core element of entire system), View (manage user interface), Controller (handle input and triggers model)
- Facade Pattern: dashboard component hide the system's complexity from the client and offers a more user-friendly interface. In this way the dashboard interface manages all the possible inputs of the client without triggering the model if it's not needed.

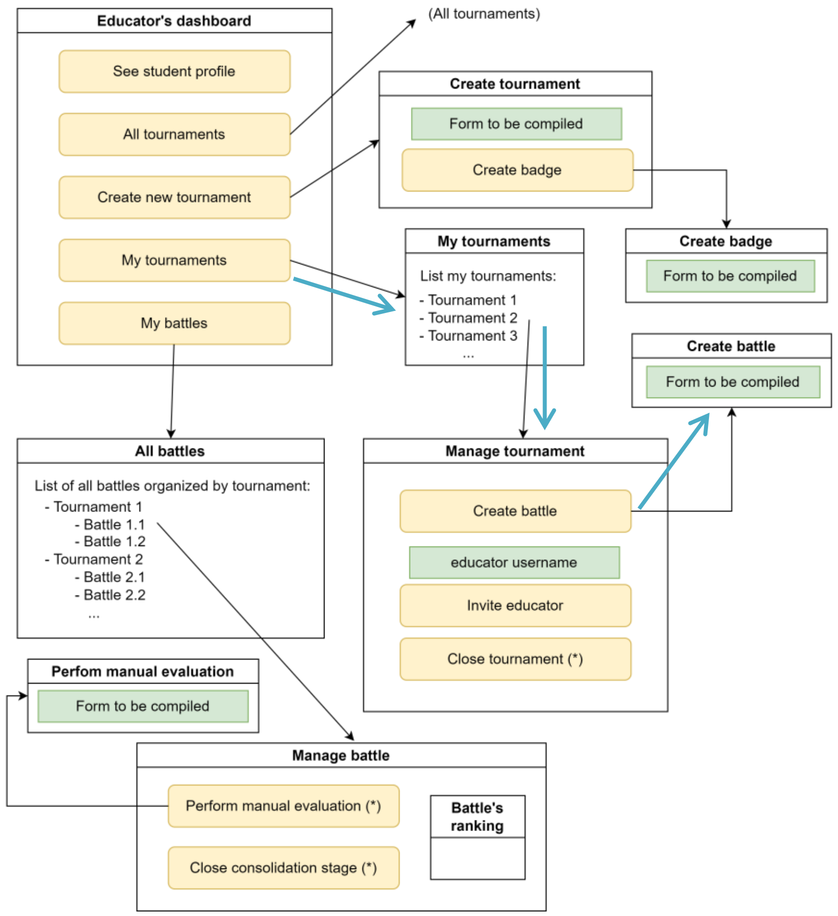




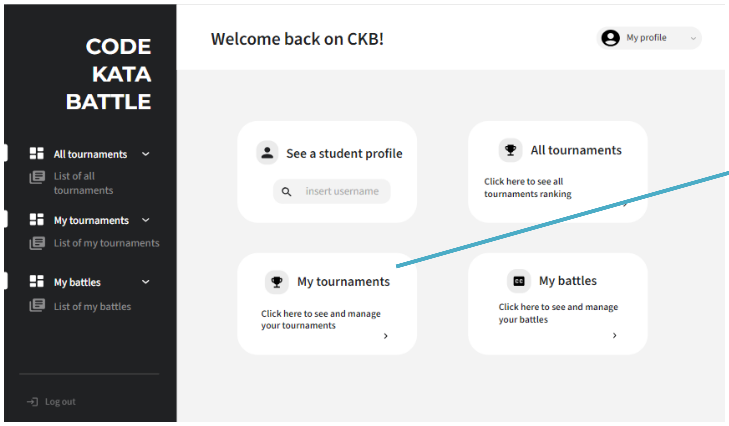
User interface design: from the RASD to the DD

Since in RASD we modelled the user interface completely clear, without any uncertainty features, so in the DD was easier to design an interface that automatically contains all the necessary features.

We designed the CKB platform as a website.
The interfaces are different according to the user who is using them, as the provided functionality are different.



User interface design: educator’s dashboard example



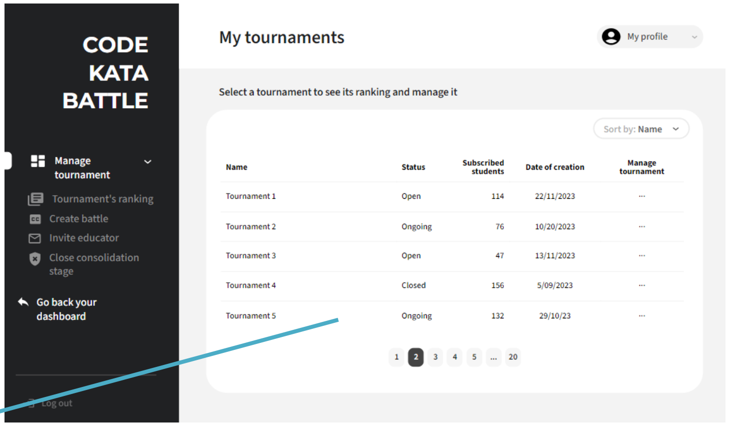
Welcome back on CKB!

See a student profile

All tournaments
Click here to see all tournaments ranking

My tournaments
Click here to see and manage your tournaments

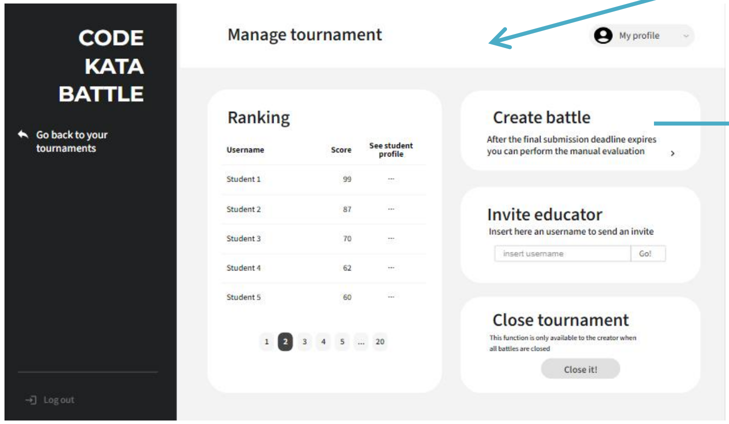
My battles
Click here to see and manage your battles



My tournaments

Select a tournament to see its ranking and manage it

Name	Status	Subscribed students	Date of creation	Manage tournament
Tournament 1	Open	114	22/11/2023	...
Tournament 2	Ongoing	76	10/20/2023	...
Tournament 3	Open	47	13/11/2023	...
Tournament 4	Closed	156	5/09/2023	...
Tournament 5	Ongoing	132	29/10/23	...



Manage tournament

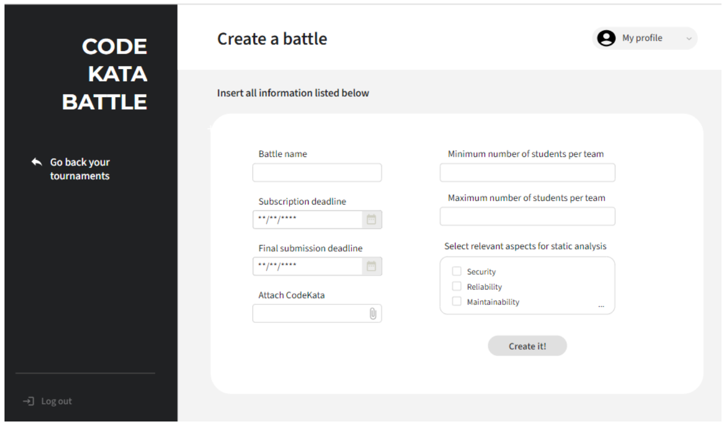
Ranking

Username	Score	See student profile
Student 1	99	...
Student 2	87	...
Student 3	70	...
Student 4	62	...
Student 5	60	...

Create battle
After the final submission deadline expires you can perform the manual evaluation

Invite educator
Insert here an username to send an invite

Close tournament
This function is only available to the creator when all battles are closed



Create a battle

Insert all information listed below

Battle name

Subscription deadline
//****

Final submission deadline
//****

Attach CodeKata

Minimum number of students per team

Maximum number of students per team

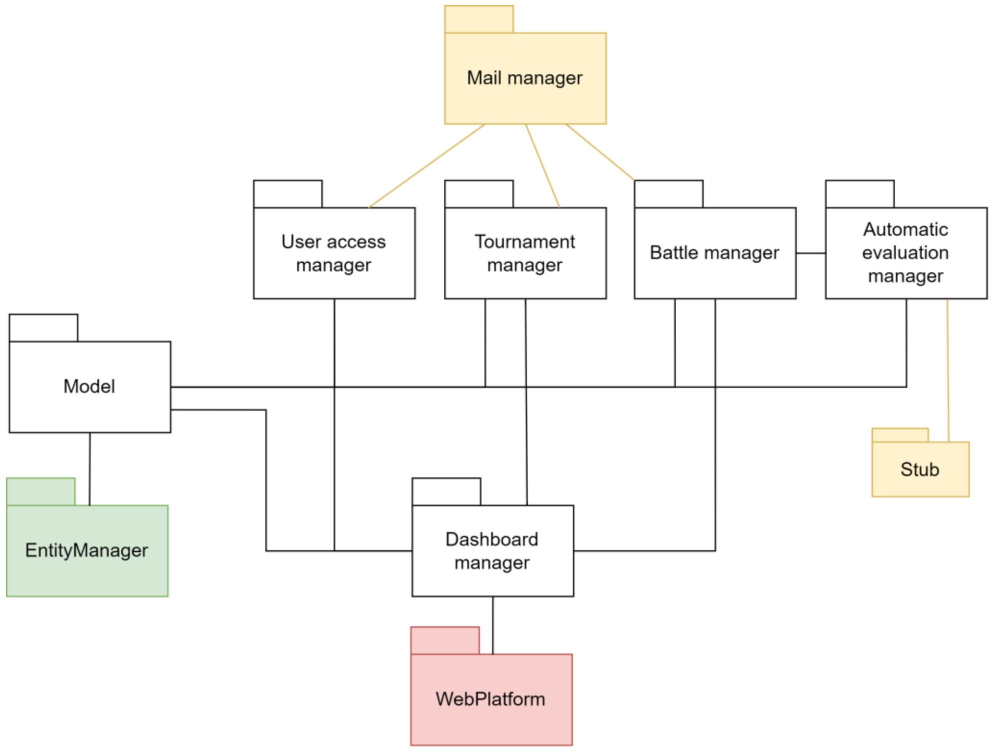
Select relevant aspects for static analysis
- ☐ Security
- ☐ Reliability
- ☐ Maintainability
...

Cohen, Gervatini, Motti

POLITECNICO MILANO 1863

Implementation, integration and test plan: development stages

1. First stage: Model and Entity Manager implemented.
2. Second stage: Add the sing up login components.
3. Third stage: Tournament manager and Battle manager components implemented.
4. Fourth stage: Automatic evaluation tool is integrated to the system
5. Fifth stage: Mail manager implemented.
6. Sixth stage: fully system is implemented.



POLITECNICO MILANO 1863

Implementation, integration and test plan: our strategy

Once the CKB system has been implemented and the singular units tested, we decided that we need to test the system as a whole in order to verify that all features works well.

Since we considered the external services as reliable, they do not need to be tested.

Initially, an alpha test of the CKB system will be conducted by the developers, to verify all functionalities using many tests and every test regard a specific feature to not be repetitive. At the alpha stage, the CKB system should be as close as possible to the final product.

Then we thought that a beta test was important. With the support of a group of educators and students they will try the web platform by their own and send a feedback to the developers, to verify the correct behavior of the system in a real environment.